你需要看看的一份前端面试题整理

作者：maixiaojie
原文链接：https://ld246.com/article/1566376065924
来源网站：链滴

最近在找工作，面了很多家，我把遇到的一些面试题整理了一下，前端的同学可以看看，也希望对正找前端工作的你有所帮助，面试题来自阿里，百度、58、好未来、便利蜂等公司。一起进步，一起成！

后续我会抽时间把这些题的答案整理出来，想看答案的同学可以期待下我后面的文章。

js

== 和 ===

当进行双等号比较时候： 先检查两个操作数数据类型，如果相同， 则进行===比较， 如果不同，愿意为你进行一次类型转换， 转换成相同类型后再进行比较， 而===比较时， 如果类型不同，直接是false.

let和const有什么区别

- const 定义的变量不能被修改，必须初始化
- let 是块级作用域，函数内部定义 let 后，对函数外部无影响
- var 定义的变量可以修改，如果不初始化，值为 undefined

数组的方法有哪些

- join() 数组元素组成字符串
- push()、pop() 数组后面添加元素、移出元素
- shift()、unshift() 数组前面删除、添加
- sort() 升序排列数组项

- reverse() 反转数组项的顺序
- concat() 将参数添加到数组中 生成新数组，不修改原数组
- slice() 截取数组，返回一个新数组，包括起始位置，不包括结束位置
- splice() 可以实现数组的修改、删除、插入。会影响原数组。第一个参数：删除起始位置，第二个数：删除个数，第三个参数：插入的元素。
- indexOf()、lastIndexOf() 从数组开头/结尾往后查找，返回索引值，未找到，返回-1
- forEach() 遍历数组，这个方法没有返回值，参数为：当前值、当前索引值、数组本身
- map() 映射，对数组每一项执行特定的函数，生成一个新的数组
- filter() 过滤，数组中每一项执行给定的函数，满足过滤条件的返回组成一个新的数组
- every() 判断数组中的每一项是否都满足条件，都满足返回 true
- some() 判断数组中是否存在满足条件的项，只要有意向满足就会返回 true
- reduce()、reduceRight() 迭代数组，参数：上一个值，当前值，当前索引，原数组； reduceRight 从最后一项开始往前进行迭代

js查找dom的方法有哪些

- 通过 id 获取（getElementById）
- 通过 name 获取（getElementByName）
- 通过标签名（getElementsByTagName）
- 通过类名（getElementsByClassName）
- 获取 html (document.documentElement)
- 获取 body (document.body)
- 通过选择器选择一个元素（document.querySelector）
- 通过选择器获取一组元素（document.querySelectorAll）

如何判断一个变量的类型

判断 Target 的类型，单单用 typeof 并无法满足。这其实并不是 bug, 本质原因是 JS 的万物皆对象理论。因此要真正完美判断时，我们需要区分对待：

- 基本类型（null）：使用 String(null)
- 基本类型（string / number /boolean / undefined）+ function: 直接使用 typeof 判断即可
- 其余引用类型（Array/ Date/ RegExp / Error）：调用 toString 后根据 [object xxx] 进行判断。

```
var class2type = {}
'Array Date RegExp Object Error'.split(' ').forEach( e => {
    class2type['[object '+e+']'] = e.toLowerCase()
})
function type(obj) {
    if(obj == null) return String(obj)
    return typeof obj === 'object' ?class2type[Object.prototype.toString.call(obj)]
    || 'object' : typeof obj;
}
```

call 和 apply 的区别

JavaScript 中的每个函数都会有 call 和 apply 的方法。

apply : 调用一个对象的方法，用另一个对象替换当前对象。

```
// A 对象调用 B 对象的方法
B.apply(A, arguments)
```

call： 调用一个对象的方法，用另一个对象替换当前对象

```
// A 对象调用 B 对象的方法
B.call(A, arg1, arg1, ... )
```

call 和 apply 不同之处在于传递的参数，apply 最多有两个参数：新 this 对象和一个数组参数，call 以参数多个参数。

手写 call

```
Function.prototype.MyCall = function(context) {
    const ctx = context || window;
    console.log(context === arguments[0]) // true
    ctx.func = this;
    var args = Array.from(arguments).slice(1);
    var res = arguments.length > 1? ctx.func(...args) : ctx.func()
    delete ctx.func;
    return res;
}
```

手写 apply

```
Function.prototype.MyApply = function(context) {
    if(typeof this !== 'function') {
        throw TypeError('Error')
    }
    var ctx = context || window;
    ctx.fn = this;
    var res = arguments.length > 1? ctx.fn(...arguments[1]) : ctx.fn()
    delete ctx.fn;
    return res;
}
```

手写promise

// TODO

手写一个深拷贝

```
function deepclone(obj) {
    if(obj == null || typeof obj != 'object') {
        return obj
    }
```

```
    let newObj = Array.isArray(obj) ? [...obj] : {...obj}
    for(let key in obj) {
        if(typeof obj[key] == 'object') {
            newObj[key] = deepclone(obj[key])
        }else {
            newObj[key] = obj[key]
        }
    }
    return newObj
}
var obj = {
    a: 1,
    b: {
        c: 2,
        d: {
            f: 5
        }
    },
    d: 5
}
console.log(deepclone(obj));
```

对异步的理解

可以从 callback、Generator、Promise、Async 和 await、定时器几个角度来进行说明。

callback:

- 嵌套函数存在耦合性，一旦有所改动，就会牵一发而动全身
- 嵌套函数一多，很难处理错误
- 不能用 try catch 捕捉错误
- 不能直接 return

Generator

Generator 是 ES6 标准引入的新的类型，一个 generator 看上去是一个函数，但可以返回多次。

原理：

他会将一个函数分成若干个小函数，并且返回一个迭代器，通过调用迭代的 next 方法，内部遍历值状态，保持正确的执行顺序。每次调用 next 方法，都会向下执行一步。

用法：

1. 先产出，下一个 next 负责传值，然后接受新值。
2. yield 后面跟的是 value 值（产出值），yield 等号前面的是我们调用当前 next 传进来的值。

注意点：

迭代器第一次 next 执行，传进来的值是无效的。

generator 一般和 promise 结合使用。可以使用 co 库。

co 库的作用：

把一个生成函数的迭代器，最后一步执行完成后，然后统一执行一个成功回调。

co 库的原理

利用产出值的 done 状态，去判断是否需要递归调用。

```
function co(it) {
    return new Promise( function(resolve, reject){
        function next(data) {
            let { value, done} = it.next(data)
            if(!done) {
                value.then(res => {
                    next(res)
                }, reject)
            }else {
                resolve(value)
            }
        }
        next()
    })
}
```

promise:

promise 翻译过来就是承诺的意思，这个承诺会在未来有一个确切的答复，并且该承诺有三种状态：

1. 等待中（pending）
2. 完成了（resolved）
3. 拒绝了（reject）

这个承诺一旦从等待状态变为其他状态就永远不能更改状态了，也就是说一旦状态变成 resolved 后就不能再次改变。

当我们在构造 promise 的时候，构造函数内部的代码是立即执行的。

promise 实现了链式调用，也就是说每次调用 then 之后返回的都是一个 promise，并且是一个全新 promise，原因也是因为状态不可变，如果你在 then 中使用了 return，那么 return 的值会被 Promise.resolved() 包装。

promise 很好地解决了回调地狱的问题。

promise的缺点：

- 无法取消 promise
- 错误需要通过回调函数来捕获

Promise.all

可以将多个 promise 实例包装成一个新的 promise 实例，同时，成功和失败的返回值是不同的，成

的时候返回一个结果数组，失败的时候返回最先 reject 的值。

```
var fun1 = new Promise((resolve, reject) => {
    setTimeout(function() {
        resolve(1)
    }, 1500);
})
var fun2 = new Promise((resolve, reject) => {
    setTimeout(function() {
        resolve(2)
    }, 1000);
})
Promise.race([fun1, fun2]).then(res => {
    console.log(res);
}).catch(err => {
    console.log(err);
})
```

promise 成功时候获取到结果数组里的顺序和 Promise.all 接受到的数组顺序是一致的。

Promise.race

哪个结果获得的快，就返回那个结果，不管状态是成功的还是失败的。

async 和 await

一个函数如果加上 async，那么这个函数就会返回一个 promise

```
async function test() {
    return 'hello'
}
console.log(test()); // Promise {<resolved>: "hello"}
```

async 就是将函数值使用 promise.resolved 进行包裹，和 then 中处理返回值一样，并且 await 只配套 async 使用。

优势：

相比直接使用 promise 来说，能够更清晰准确的写出代码，优雅的解决了回调地狱的问题。

缺点：

如果多个异步代码没有依赖性却使用了 await,会导致性能上的降低。

定时器

setTimeout、setInterval、requestAnimationFrame(自带节流功能)

es6里用过哪些新特性

可以从 promise/generator 等角度出发，引申出异步处理相关的知识点；

可以从 let const 等角度出发，引申出变量提升、词法分析、原型链、继承等相关知识点。

可以从 Map/Set 等新的数据类型出发，引申到类型的检查、数组的一些方法、ts 等。

promise和callback的区别

详见上面对异步的理解。

说下promise

详见上面对异步的理解。

说一下闭包

闭包是由函数和创建该函数的词法环境结合而成的，这个环境包含了这个闭包创建时所能访问到的所局部变量。

```
function makeFunc() {
    var name = "Mozilla";
    function displayName() {
        alert(name);
    }
    return displayName;
}

var myFunc = makeFunc();
myFunc();
```

说一下原型链

每个实例对象都有一个私有属性（称之为 __proto__），指向它的构造函数的原型对象（prototype），该原型对象也有自己的原型对象（__proto__），层层向上，直到一个对象的原型对象为 null。根定义，null 没有原型，并作为这个原型链中的最后一环。

说一下原生ajax的实现

```
var xhr = new XMLHttpReuest();
xhr.onreadystatechange = function() {
    if(xhr.readyState == 4) {
        if(xhr.status == 200) {
            alert(xhr.responseText)
        }
    }
}
xhr.open('GET', '/api', false);
xhr.send();
```

什么是事件代理

```
var ul = document.getElementById('parentUl'),
        li = ul.getElementsByTagName('li');
    function addClick() {
```

```
        for (var i = 0; i<li.length;i++){
            li[i].onclick=function () {
                alert(this.innerHTML);
            }
        }
    }
    addClick();
    function addElement() {
        var li = document.createElement('li');
        li.innerHTML="我是新孩子";
        ul.appendChild(li);
        addClick();
    }
```

数组扁平化怎么实现

flat

```
var arr = [1, [2, 3, [4, 5]], [6, 7], '8'];
console.log(arr.flat(Infinity));
```

有兼容性问题，暂时不推荐使用。

使用 reduce

```
var arr = [1, [2, 3, [4, 5]], [6, 7], '8'];
function flat(arr) {
    if(!Array.isArray(arr)) {
        return arr;
    }
    return arr.reduce((pre, cur) => {
        return pre.concat(flat(cur))
    }, [])
}
console.log(flat(arr))
```

toString

```
var arr = [1, [2, 3, [4, 5]], [6, 7], '8'];
console.log(arr.toString().split(',').map(Number))
```

适用于数组的元素都是数字。

concat

```
function flat1(arr) {
    return !Array.isArray(arr) ? arr : [].concat.apply([], arr.map(flat1));
}
console.log(flat1(arr))
```

reduce的参数

Object.freeze()的作用

v8的垃圾回收机制

浏览器事件循环是怎样的

浏览器和node中事件循环有什么不一样的

把一个数组随机打乱

```
var arr = [1, 23, 4, 5, 6, 7, 8, 9]
var newArr = arr.sort(function(a,b) {
    return Math.random() - 0.5
})
console.log(newArr)
```

节流如何实现

用setTimeout实现一个setInterval

```
function mysetInterval(fn, after, cancle) {
    var timer = null;
    if (cancle) {
        clearTimeout(timer);
    } else {
        timer = setTimeout(function() {
            fn();
            mysetInterval(fn, after, cancle)
        }, after);
    }
}
mysetInterval(function() {
    console.log(1);
}, 1000, false)
```

canvas 拖拽放大缩小，如何实现

思路： 监听鼠标事件，对相应的 dom 宽高进行变化，注意边界的问题

CSS

css动画

浏览器兼容性怎么处理

css选择器有哪些

雪碧图怎么实现

怎么居中

flex布局是怎么做的

less有哪些特性

vue

vue的优缺点

// TODO

vue和其他框架的对比

// TODO

实现一个双向绑定

使用 Object.defineProperty , 在 get 和 set 方法中做相应的处理。

vue生命周期

- beforeCreate
- create
- beforeMounted
- mounted
- beforeUpdate
- updated
- beforeDestroy
- destroy

vue-router原理

- HashHistory

hashHistory.replace() 和 hashHishtroy.push()

监听地址变化 window.onhashchange

- HTML5History

window.history.pushState() 和 window.history.replaceState()

监听地址变化 window.onpopstate

vue组件如何通讯

- 父子组件

props $emit

- provide inject
- 全局状态管理（vuex/eventBus等）

说一下vuex

从 store、getters、mutations、actions、modules 等核心概念出发，

可以说说数据如何流转、mutation 和 action 的区别等。

vuex的原理

vuex 仅仅是作为 vue 的一个插件而存在。

每个 vue 插件都有一个公开的 install 方法，vuex 也不例外，调用了 applyMixin 方法，在所有组件 eforeCreate 生命周期中设置了 this.$store 这样一个对象。

其本质就是将我们传入的 state 作为一个隐藏的 vue 组件 的 data，我们的 commit 操作，本质上修这个组件的 data，修改被 defineReactive 代理的对象值后，会将其收集到依赖的 watcher 中的 dirty 设置为 true，下一次重新访问该 watcher 的时候会重新获取最新值。

这样就能解释的清楚为什么 vuex 中 state 的对象数组必须前提定义好。

vuex中的store本质就是没有template的隐藏着的vue组件

vue和jq有什么区别

vue源码看过哪些（阿里）

vdom为什么比dom快?

webpack

webpack是干嘛的

webpack里的插件和loader有什么区别

说下你使用过webpack里的插件和loader

如何写一个webpack插件或者loader

webpack打包怎么优化

sourcemap

设计模式

单例

发布订阅

http

http1 http2 https有什么区别

http缓存机制

TCP和UDP有什么区别

控制缓存的字段有哪些

你知道的长链接有哪些

nodejs

nodejs做过哪些东西

nodejs用过哪些模块

express和koa区别

算法

排序算法

```
var arr = [1, 3, 2, 43, 23, 31, 56, 21, 22];
function checkArray(arr) {
    return Array.isArray(arr)
}
function swap(arr, pre, aft) {
    var temp = arr[pre];
    arr[pre] = arr[aft];
    arr[aft] = temp;
```

```
}
// 冒泡排序
function bubble(arr) {
    if(!checkArray(arr)) return
    for(var i=arr.length-1; i>0; i--) {
        for(var j=0; j<i; j++) {
            if(arr[j] > arr[j+1]) {
                swap(arr, j, j+1)
            }
        }
    }
    return arr
}
// 插入排序
function insertion(arr) {
    if(!checkArray(arr)) return
    for(var i=1; i<arr.length-1; i++) {
        for(var j=i-1;  j>0 && arr[j] > arr[j+1]; j--) {
            swap(arr, j, j+1)
        }
    }
    return arr
}
console.log(bubble(arr));
console.log(insertion(arr))
// 冒泡排序
function bubb(arr) {
    var len = arr.length;
    for(var i=len; i>0; i--) {
        for(var j=0; j<len-1-i; j++) {
            if(arr[j] > arr[j+1]) {
                [arr[j], arr[j+1]] = [arr[j+1], arr[j]]
            }
        }
    }
    return arr
}
console.log(bubb(arr))
// 快速排序
function quicksort(arr) {
    if(arr.length <=1) {
        return arr;
    }
    var pivot = arr.splice(Math.floor(arr.length/2), 1)[0];
    var left = [], right = [];
    for(var i=0; i<arr.length; i++) {
        if(arr[i] < pivot) {
            left.push(arr[i])
        }else {
            right.push(arr[i])
        }
    }
    return quicksort(left).concat([pivot], quicksort(right))
}
```

```
console.log(quicksort(arr))
```

手写快排

如上。

手写冒泡

如上。

用两个队列实现一个栈

思路： 用其中一个栈来做数据中转站。

数组找最大值

思路1：

排序后拿末端的值

思路2：

逐个比较

其他

小程序的生命周期

小程序路由跳转有哪些方式

小程序页面之间如何传值以及接受值

对jsx的理解

js大数精度问题如何处理

serviceworker是干嘛的

web优化有哪些方法

linux熟悉程度

印象最深的bug

项目中遇到最难解决的问题

聊聊你参与过的一个项目

前端项目如何部署

如何封装一个组件

说一下弹框的设计思路

微服务和serverless有了解过吗